Embedded System Design

CSEE 4840 - Course Project

♪ The Notator ♫

(a.k.a. the B flat B sharp pitch detector)

By

Andrew Melkonian
James Rishe
Narashiman Chakravarthy

Abstract

We have designed a system to convert a human-produced musical symbol into pitch intervals that could be displayed on a computer screen, or conceivably in some sort of memory. Audio data samples are taken by the audio codec at 48 kHz and stored in an audio buffer. From the amplitude data in the buffer the frequency of the audio input is determined using a fast Fourier transform (FFT) algorithm coded in C. This frequency data is then broken down into different pitch ranges and outputted to the screen through the VGA controller.

The applicability of this design would be in some kind of portable device that would allow musicians to sing into it as they are going from place to place so that they can make a note of some tune that they have devised in their head so that they can transfer it to music paper or some other form of notation at a later time. Even as a non portable device, there is plenty of value in being able to directly produce notes and musical intervals from an audio signal.

Table of Contents

1. Overview and Design

The overall project is to convert a human produced musical note or a signal and display the fundamental frequency value or frequency range on the computer screen or store it in a device. A microphone collects the audio signal and the signal is converted into digital signal by the board elements. The Digital signal is sampled by the audio codec and stores it in a buffer (audio buffer). Now a C program accesses the buffer and FFTs the samples and outputs the fundamental frequency or the frequency range of the signal samples. This range is displayed in the VGA as output.

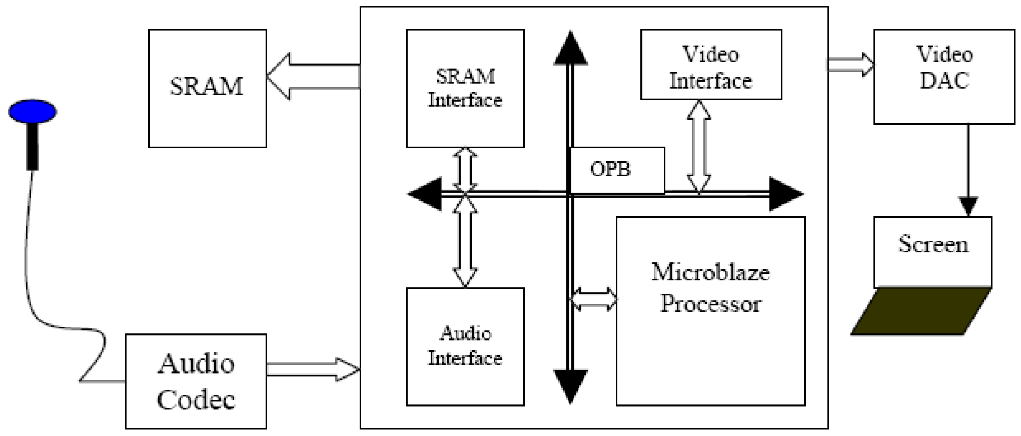

Diagram 1. Original block diagram of the final project

Notator uses the SpartanII-E FPGA on the XSB-300E Board with the on-board audio and video peripherals. The on-board audio input accepts an external analog stereo line-level signal through the 1/8” mini jack. The signal undergoes Analog-to-Digital conversion with the assistance of the on-board AK4565 (audio codec). A Fast Fourier Transform is applied in order to appropriately isolate independent ranges in the frequency domain. The results of the FFT are pipelined into a graphics visualization function and then sent to the video hardware to produce a proper visualization. The end result is a Frequency Range display of the corresponding signal.

2. What we Intended to do

In our original project abstract, we intended to display the individual notes. Unfortunately since our desired bands are based on the exponential scale, for lower order of Frequencies to be distinct, FFT precision must be in higher. That means the initial slot of the FFT would decide many of the frequency ranges. Notes in this case and final frequency will have many FFT slots, which is a clear mismatch. We had to calculate and decide upon what we had to do in terms of the FFT bins and frequency ranges in order to solve this issue.

3. What we actually did

In the end, we wound up using an FFT with 1024 buckets, which by nature are doubled, so each FFT execution yielded 512 buckets of unique information. These are linearly spaced between 0 Hz and half of the sampling frequency, which is 48 kHz, so half is 24 kHz. Dividing 24,000 by 512, we find that each bucket corresponds to a range of about 46.9 Hz. This granularity is too coarse to distinguish individual notes up to about a pitch of 880 Hz (high A). However, much more significant was that the maximum amplitudes that we extracted from the FFT were highly erratic. As a result, we were not able to split the sound spectrum up into individual notes, but could only divide it up into six different ranges of pitches with any degree of confidence.

We chose these ranges to increase exponentially in size for higher pitches, because human hearing judges pitch distance logarithmically. The following table gives the insight into how the frequency ranges are divided and distributed.

Range	FFT Buckets	Frequency Range (Hz)
No Sound	0-1	0-70.3
Low	2-4	70.3-210.9
Low-mid	5-10	210.9-492.2
Mid	11-23	492.2-1101.6
Mid-high	24-47	1101.6-2226.6
High	48-95	2226.6-4476.6
Very High	96-169	4476.6-7945.3

With this scheme, we were able to get fairly consistent ranges for a constant sound and were able to show which pitch range a sound was in without too much variation. Although we did not accomplish what we initially set out to do, we did manage to use the XSB board to do Fourier transform and pitch detection, and consider that a small victory.

4. Design components

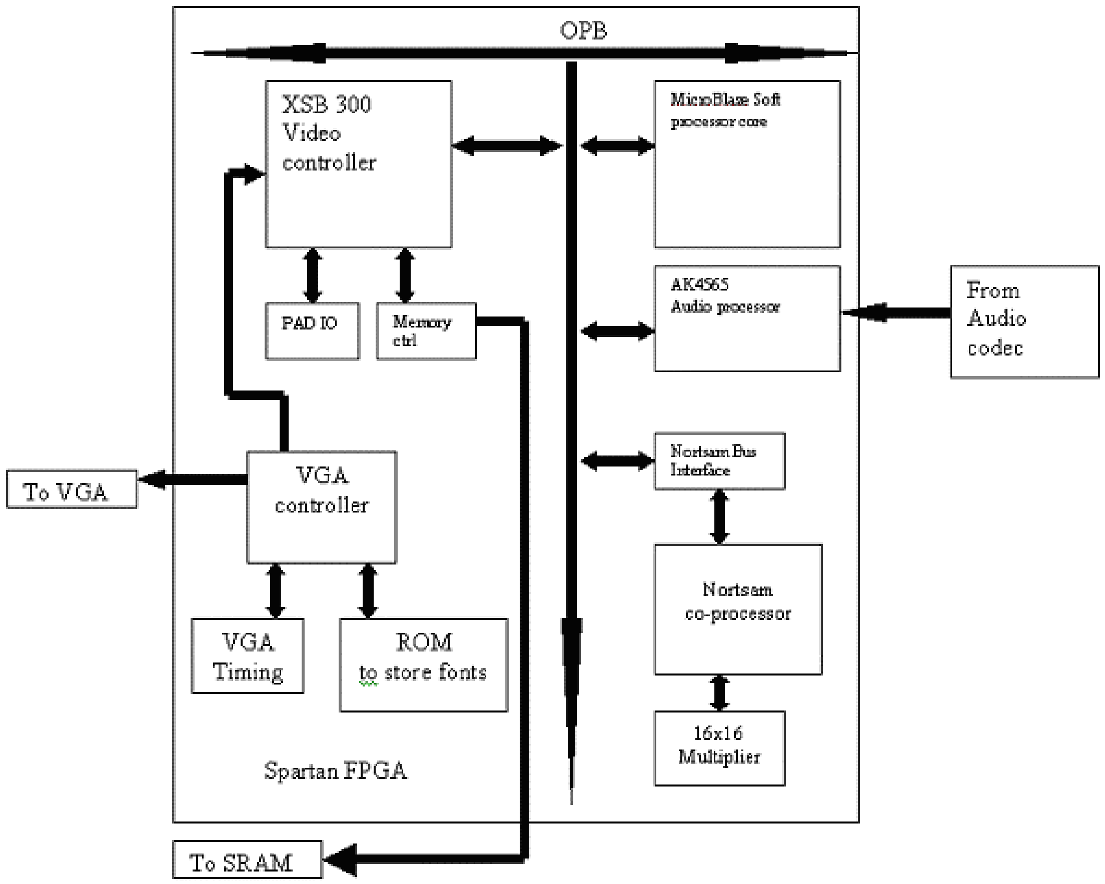

Diagram 2: Overall Block Diagram of the final project

4.1 Hardware Description

The Block diagram above shows the components available in the final design. Of those available we used the Nortsam co processor and its peripheral 16x16 bit multiplier. WE made major modifications in the way multiplier multiplies the two 16 bit numbers, which is explained later in the report. We did modify the co processor for our needs. But major portion of the time for the project was spent on the following two components.

4.2 VGA Controller

This peripheral performs three important functions. First, it controls the VGA by generating the necessary timing signals (horizontal sync, vertical sync, and blanking, which blanks the video output during horizontal and vertical refresh), and memory addresses for each pixel, and eventually formats and ships out this video data to the triple 8-bit video DAC on the board. Second, it controls the off-chip SRAM through address and data busses, chip select signals and other means. The glue logic in the main module, opb_xsb300.vhd, which generates the signals and the off-chip drivers for these signals are in pad_io.vhd. It arbitrates access to this memory between the video controller and the processor. In each cycle, the processor, video, or both may want access to the memory. Since the video absolutely needs the memory when it asks for it the memory controller gives priority to the video system, possibly making the processor wait its turn.

Since this VGA controller was used for text, the values actually stored in memory were the ASCII values of the characters of the screen, and a font ROM was used to store the fonts. We slightly modified the font ROM to contain flat (♭) and sharp (♯) characters, which were mapped to the original carat (^) and pound (#) ASCII values.

4.3 Audio CoDec (AK4565)

This audio CoDec was originally implemented by Christian Soviani. We made slight modifications to the original code with Marcio's help because the code we had imported various Unisim libraries, and declared and used BRAMs in separate VHDL files. The audio CoDec's work in the system is to generate a bit-stream of data synchronized to a clock signal generated by it. The three clock signals which are generated (Master, serial and Frame) are 12.5 MHz, 1.56MHz, and ~49 KHz. The bit-stream which is created is fed into a BRAM module to buffer it until there are 64 samples. Once the buffer is full, an interrupt is sent to the processor, and the processor has the task of either getting the contents of the buffer or ignoring the interrupt.

The interface is memory-mapped to base address 0xFEFE0000 on the OPB bus. When a write is performed from software (in support of audio output), the interface prefetches the incoming sample at the corresponding address. A 32-bit value is returned upon the next read to the interface's address space, representing one stereo amplitude sample at Δt=(1/48.828khz). A 48 kHz sampling rate is used because human hearing extends to approximately 20 kHz, and Nyquist's sampling theorem states that a signal with a frequency bandwidth N must be sampled at at least 2N in order to be able to reconstruct the original signal entirely.

4.4. The Nortsam Multiplication unit

The group Nortsam from last year put together a great component that tremendously helped us in our project. It consisted of four 16x16 multipliers and code for arbitrating between the OPB and the multipliers. It was capable of operating in two modes, depending on which write addresses were used:

In the first mode, the multiplier took in two 16-bit complex numbers in the form of two 32-bit unsigned integers. Each 16-bit component was from an audio sample and ranged between -2^{15} and $2^{15}-1$, to represent a number between -.5 and $.5-(1/2^{15})$. It multiplied the numbers together in parallel (complex multiplication involves four real multiplications) and returned the result as a 32-bit unsigned integer with the two halves representing the result of the operation. This was a fixed-point multiplier which meant that the result was also a number between -2^{15} and $2^{15}-1$ that represented the result, between -.5 and $.5-(1/2^{15})$. In other words, the result is in the same format as the input. When we tested this part, we found that it worked, but that the results did not have all 16-bits of accuracy. We modified it and improved it as described in section 5.7.

The other mode took in two 16-bit numbers as a 32-bit integer, squared both of them, and returned the results, formatted so that the decimal point in the result is twice as far over in the result as it is in the input. We did not observe any incorrect results from this mode, and did not change it.

4.5. FFT and Its Importance

Fast Fourier Transform (FFT) is a type of Discrete Fourier transform (DFT), but only faster with fewer computations (summations and Multiplications). A DFT takes N^2 computations to calculate a transform for N points, while the FFT takes around Nlog2N computations to complete the same thing. The input array contains *N* (a power of 2) complex time samples in a real array of length 2*N* (since the samples are real, the complex part is set to 0), with real and imaginary parts alternating in the array. The output array contains the complex Fourier spectrum at *N* values of frequency. Real and imaginary parts again alternate. The output array starts with zero frequency, works up to the most positive frequency (which is ambiguous with the most negative frequency, and is half the sampling frequency, so 24 kHz). Negative frequencies follow, from the second-most negative up to the frequency just below zero.

The algorithm contains two parts. The first section sorts the data into bit reversed order. That is because when calculating the FFT butterfly, reverse of 1 is 100 which is 4, and reversal of 2 is 010 which is 2 again, reversal of 3 is 110 which is 6, etc., to get the frequency components in order from smallest to highest. The second part contains the transform part which calculates the transforms in length 2,4,6 ..N. For each stage of this process, two nested inner loops range over the subtransforms already computed and the elements of each transform, implementing the Danielson-Lanczos Lemma.

We implemented the fast Fourier transform (FFT) code in C. We wrote our own bit reverse and separate functions. For the actual transformation we used a modified version of the algorithm from C^2 Numerical Algorithms, which the Nortsam group also used last year, and so we got a very similar result to theirs. We do not have to work with the right audio channel because the microphone is monaural.

5. The Steps of our project, from start to finish

Along the way, we encountered many obstacles in the way of reaching a working project. These were some of the major milestones that we had, why we encountered them, and how we resolved them.

5.1 Working VGA controller

Getting the VGA to work turned out to be unexpectedly one of the hardest parts of the project, on account of our use of the audio codec and our moderately sized C programs. The main obstacles that we had to overcome for this step were the limited number of BRAMs available in the microblaze. At the start of the project, we placed the Lab 2 VGA controller into the lab 6 project, figuring a design with a VGA controller and an SRAM was a good start. This was fine until we tried to incorporate the pre-made audio codec interface into our project. Since the audio controller and the VGA controller both used BRAMs, the combined amount was too much for the board to handle. We tried to figure out how the fonts were stored in the VGA controller, but it seemed far too confusing at the time. We had to seek other means.

Next we tried the VGA controller from the "game" group from last year, which used no BRAMs at all. We figured out how their font RAM worked, and made our own custom font, but when it came time to try out the VGA controller, it would not work for us. We tried for about a week and a half to get it to work, but with no luck, we found that we had to go back to the drawing board again.

With a considerably improved understanding of fonts, we went back to the Lab 2 VGA controller and tried to modify it. We removed the font BRAMs from the module and replaced the font with a custom-made ROM, modifying the code to correctly access it. The project compiled, the VGA controller worked. It seemed that our problems were over. However, as soon as our C code grew slightly large, it no longer fit on the board, since there still were not enough spare BRAMs.

Help came in the form of a completed Lab 5 project that Marcio had kept from last year. This design used methods for storing some of the C instructions in the SRAM, giving us much more space. It also gave us a ready-to-go VGA controller. We used that for the new basis of our project, and after some wrestling with the audio codec (see next section), it was up and running.

5.2.1 Working audio codec (getting meaningful samples)

The major problem that we encountered when getting a working audio codec was a cryptic error that we were receiving during compilation. After we struggled with it for a while, Marcio took a look, and found that a strange Unisim declaration was at the top of the files and furthermore, that a BRAM was declared in one .vhd file and used in another. After fixing this, the project compiled.

5.2.2 Using the audio data

We used last year's Nortsam project as a jumping off point for the rest of the project. We soon found that we could not get their FFT code to run without crashing, so we decided to start from the beginning. We found their code to be extremely cryptic. They sometimes used 32 bit integer arrays as 16 bit integer arrays with twice as many elements, and parts of their code tried to access array elements past the end of the array. We put the FFT on the backburner and tried to figure out how the codec worked. We found that it sent a stream of 32-bit integers, of which the first 16 bits represented a sample from the left channel, and the next 16 bits, a sample from the right channel. We set about peeling off these values for use in the FFT. The Nortsam group tried to do this separation at the same time as the beginning of the FFT, with seemingly disastrous results.

When we first thought about how to get an audio signal from the microphone we thought that we were supposed to plug the microphone into the blue jack in the audio section of the board (typically line in), since we had read in project reports from last year that trying to access the microphone input jack was problematic. After several hours, we realized, as other groups had, that the pins to the right of the jack were in fact the place where we had to connect our audio input device once we hooked up cables to accomplish the connection, we started getting data from the microphone.

5.3 Doing first half of the FFT (separate and swap functions)

As just mentioned, the Nortsam group from last year had tried to combine separating the sample data performing the swap part of the FFT into one function. We found this to be far more confusing than it merited, and possibly not more useful than doing the two operations separately. In this combined function they used an array of 32-bit numbers as an array of 16-bit numbers with twice as many elements. We were not able to run their code without crashing, and we decided to keep it simple and just use an array of actual 16 bit integers, and perform the two procedures separately. We obtained the algorithm from C^2 Numerical Recipes from Cambridge University press, as they had, and slightly modified it for our purposes. We were now ready to move onto the bulk of the FFT operation.

5.4 Doing the primary part of the FFT (doFFT)

This procedure was the second half of the algorithm from Numerical Recipes, and the part that required the complex multiplications. The Nortsam group had created a VHDL component of the same name, which did 16-bit fixed-point complex multiplications. We tested the component, and found that the results were inaccurate in the two or three least significant bits, but essentially correct. We incorporated it into the algorithm. At this point we struggled with the C compiler which erratically gave cryptic errors about assembly code. We suspect that this was the result of trying to do too many operations within one loop. We took a lesson from Nortsam yet again and this time accessed the 32-bit result of a memory read as an array of two 16-bit integers, and this was seemingly just enough to get the compiler to stop complaining.

5.5 Getting maximum magnitude of FFT

Once we had the FFT running, we used the Nortsam component's other functionality – a dual floating-point squarer, to find the magnitudes of the buckets resulting from the FFT. After that, we used a simple max function, weighting towards nonzero results, and essentially our project was near completion. Since we found the output of the FFT to be somewhat erratic, we collected maxima for 10 FFT operations, and then went with the range that had had the most hits in that time.

5.6 Formatting and displaying data from FFT on VGA

We broke the magnitudes down into seven different pitch ranges, as described in section 2.2. We have several functions that display this data on the screen by showing a note symbol next to the pitch range of the sound that the audio input device reads. We also modified the font ram to display sharp and flat signs, since it seemed like a waste to have struggled so long with fonts, and not make anything of the experience.

5.7 Improving the 16-bit fixed point multiplier

In an effort to improve the results of our FFT, we set about to improve the Nortsam component, which we had earlier found to be inaccurate. Initially, this component multiplied two complex numbers with 16-bit components and returned one complex number, also with two 16-bit components. The problem was that it used 16-bit intermediate values, and rounding along the way, caused the resulting product to have less than 16 bits of accuracy. After we figured out how the component worked, we tried making a 17-bit multiplier for slightly higher precision. This multiplier did work, but it slowed the clock down from 18.1 ms to 26.4 ms. We figured that it was probably doing 32-bit addition and we decided that as long as it was going to do the extra time, we may as well do the intermediate multiplication using 32-bit intermediate values and truncate at the very end. To our great surprise, this actually sped the clock up to 17.9 ms and seemingly improved the results of our algorithm — a win on both fronts.

6. What We Learned

6.1 Andrew K. Melkonian

Nailing down a solid project idea early is very important. This allows a group to get a head start on implementing their project and figuring out VHDL and the microblaze. The group can determine early on which goals are realistic within the provided timeframe and which goals are simply not feasible. Also, I learned that having a project in an area that one is familiar makes the project more interesting and allows one to contribute more to the project. I felt that if I was more familiar with music then I would have been able to contribute more to the project in terms of implementing the FFT.

6.2 Narashiman Chakravarthy

I had a lot to learn from this project and did learn some, but the course overall has taught me more than this project did alone. The idea of OPB and PLB in the Micro blaze/FPGA and the idea of the user IP and implementation of user peripheral is going to be a lot helpful for my future endeavors. One such future is my other project in CCNY which I will be implementing a IP in a similar board, only bigger. This course has given me strong foundation for that. Now one more aspect the project has taught me is the group work, and what the project mates are for. They help you when you slip both in terms of understanding the concepts and manual stuff like clearing up doubts in the Codes. If only I was more familiar with the C code might have been able to contribute more and produce a more dashing project.

6.3 James Rishe

This semester I studied a lot about digital audio signal processing in my Music Signal Processing class. It was a good experience to work on this project and truly deal with actual audio samples from an audio codec. I also learned a lot about how fonts are represented on systems such as these and generally in most text-based environments. Spending long hours sifting through VHDL files trying to figure out how they worked and how to improve them greatly augmented my experience with VHDL and I feel like I know enough now to actually go into the workforce and work on embedded systems. For that I am very grateful to professor Edwards and Marcio Buss.

7. Who did what

Andrew Melkonian

Graphical display, FFT, Nortsam multiplier improvement, fonts, error debugging

James Rishe

Original design, fonts, FFT, graphical display, working with audio codec, Nortsam multiplier improvement

Narashiman Chakravarthy

Displaying design layouts in CAD tools, FFT logistics, Nortsam multiplier improvement

8. Advice for Future Teams

The best piece of advice we, or possibly anyone, can provide for future teams is start early. Getting an early start on a project gives a team the best chance to achieve their goals. It also allows a team to make their project cool and interesting rather than just making it work. Unfortunately we did not start early as we had some issues as to what are project was actually going to be.

Another piece of advice we can give future teams is to pick (if possible) a project that draws on the skills of all the group members and that interests all the group members. This ensures that every group member is able and willing to participate fully in the project.

Lastly, make sure you go to the lab often, especially when the TA is there. The TA can help a group understand cryptic error messages and resolve problems with the code so that the group can focus on making the actual ‘meat’ of their project rather than spending too much time staring at the screen (though spending some time staring at the screen is certainly a necessary part of the learning experience)

9. VHDL Code

--Filename: audio_ak4565.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--library UNISIM;
--use UNISIM.VComponents.all;


entity audio_ak4565 is

port  (
         CLK : in std_logic;
            RST : in std_logic;

            audio_bram_clk : out std_logic;
            audio_bram_addr_dac : out std_logic_vector(11 downto 0);
            audio_bram_addr_adc : out std_logic_vector(11 downto 0);

            audio_bram_dacdata : in std_logic;
            audio_bram_adcdata : out std_logic;

            interrupt : out std_logic;
            audio_fifohalf : out std_logic;

            AU_CSN_N : out std_logic;
      AU_BCLK : out std_logic;
      AU_MCLK : out std_logic;
      AU_LRCK : out std_logic;
      AU_SDTI : out std_logic;
      AU_SDTO0 : in std_logic
        );
end audio_ak4565;


architecture Behavioral of audio_ak4565 is


signal clkcnt : std_logic_vector(4 downto 0);
signal audiocnt_dac, audiocnt_adc : std_logic_vector(11 downto 0);

signal audio_clk : std_logic;
signal lrck : std_logic;

begin

AU_CSN_N <= '1'; -- no chip select as we don't write the ctrl regs

-- generate the 3 clocks: master, serial, frame

process(CLK, RST)
begin
```

```
      if rst = '1' then
            clkcnt <= "00000";
      elsif clk'event and clk='1' then
            clkcnt <= clkcnt + 1;
      end if;
end process;
AU_MCLK <= clkcnt(1); -- master clock, 12.5 MHz
audio_clk <= clkcnt(4); -- this is the serial clock, 1.5625 Mhz
AU_BCLK <= not audio_clk; -- dont't ask but read AK4565 specs

process(audio_clk, RST)
begin
      if rst = '1' then
      audiocnt_dac <= "000000000000";
      audiocnt_adc <= "111111111111";
      lrck <= '0';
      elsif audio_clk'event and audio_clk='1' then
            audiocnt_dac <= audiocnt_dac + 1;
            audiocnt_adc <= audiocnt_adc + 1;
            lrck <= not audiocnt_dac(4);
      end if;
end process;
AU_LRCK <= lrck; -- audio clock, 48.828 kHz

interrupt <= not audiocnt_dac(10);
audio_fifohalf <= audiocnt_dac(11);

audio_bram_addr_dac <= audiocnt_dac;
audio_bram_addr_adc <= audiocnt_adc;
audio_bram_clk <= audio_clk;

AU_SDTI <= audio_bram_dacdata;
audio_bram_adcdata <= AU_SDTO0;

end architecture;
```

--Filename: opb_xsb300_ak4565.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

--library UNISIM;
--use UNISIM.VComponents.all;

entity opb_xsb300_ak4565 is
  generic (
    C_OPB_AWIDTH    : integer := 32;
    C_OPB_DWIDTH    : integer := 32;
    C_BASEADDR      : std_logic_vector(31 downto 0) := X"FEFE_0000";
    C_HIGHADDR      : std_logic_vector(31 downto 0) := X"FEFE_FFFF"
    );

    Port (  OPB_Clk : in std_logic;
                      OPB_Rst : in std_logic;
                      OPB_ABus : in std_logic_vector (31 downto 0);
                      OPB_BE : in std_logic_vector (3 downto 0);
                      OPB_DBus : in std_logic_vector (31 downto 0);
                      OPB_RNW : in std_logic;
                      OPB_select : in std_logic;
                      OPB_seqAddr : in std_logic;

                      UIO_DBus : out std_logic_vector (31 downto 0);
                      UIO_errAck : out std_logic;
                      UIO_retry : out std_logic;
                      UIO_toutSup : out std_logic;
                      UIO_xferAck : out std_logic;

                              ak4565_intr : out std_logic;
Interrupt : out std_logic;

            AU_CSN_N : out std_logic;
            AU_BCLK : out std_logic;
            AU_MCLK : out std_logic;
            AU_LRCK : out std_logic;
            AU_SDTI : out std_logic;
            AU_SDTO0 : in std_logic
                     );
end opb_xsb300_ak4565;

architecture Behavioral of opb_xsb300_ak4565 is

component audio_ak4565 port   (
         CLK : in std_logic;
            RST : in std_logic;

            audio_bram_clk : out std_logic;
            audio_bram_addr_dac : out std_logic_vector(11 downto 0);
            audio_bram_addr_adc : out std_logic_vector(11 downto 0);

            audio_bram_dacdata : in std_logic;
            audio_bram_adcdata : out std_logic;

            interrupt : out std_logic;
            audio_fifohalf : out std_logic;

            AU_CSN_N : out std_logic;
      AU_BCLK : out std_logic;
      AU_MCLK : out std_logic;
      AU_LRCK : out std_logic;
      AU_SDTI : out std_logic;
      AU_SDTO0 : in std_logic
        );



end component;


component RAMB4_S1_S16
--

  port (DIA    : in STD_LOGIC_VECTOR (0 downto 0);
        DIB    : in STD_LOGIC_VECTOR (15 downto 0);
        ENA    : in STD_logic;
        ENB    : in STD_logic;
        WEA    : in STD_logic;
        WEB    : in STD_logic;
        RSTA   : in STD_logic;
        RSTB   : in STD_logic;
        CLKA   : in STD_logic;
        CLKB   : in STD_logic;
        ADDRA  : in STD_LOGIC_VECTOR (11 downto 0);
        ADDRB  : in STD_LOGIC_VECTOR (7 downto 0);
        DOA    : out STD_LOGIC_VECTOR (0 downto 0);
        DOB    : out STD_LOGIC_VECTOR (15 downto 0));
end component;

signal cs, xfer, xfer_1, xfer_2 : std_logic;

signal rnw : std_logic;
signal addr : std_logic_vector (15 downto 0);
signal wdata : std_logic_vector (31 downto 0);

signal rdata : std_logic_vector (31 downto 0);
signal opb_ad_ce : std_logic;

signal we, a0, ce0, ce1 : std_logic;
signal bram_rdata, bram_wdata : std_logic_vector(15 downto 0); -- as
the CPU sees them
signal bram_rdata_rev, bram_wdata_rev : std_logic_vector(15 downto 0);
-- reversed !
signal bram_addr : std_logic_vector(7 downto 0);

signal audio_bram_clk : std_logic;
signal audio_bram_addr_dac, audio_bram_addr_adc : std_logic_vector(11
downto 0);
signal audio_bram_dacdata : std_logic;
signal audio_bram_adcdata : std_logic;
signal audio_bram_dacdata_v : std_logic_vector(0 downto 0);
signal audio_bram_adcdata_v : std_logic_vector(0 downto 0);

signal audio_fifohalf : std_logic;

begin

process(OPB_Rst, OPB_Clk)
begin

-- register adresses, data write, rnw
   if OPB_Rst = '1' then
          addr <=  X"0000";
          wdata <= X"0000_0000";
          rnw <= '0';
      elsif OPB_Clk'event and OPB_Clk ='1' then
          if opb_ad_ce = '1' then
                         wdata <= OPB_DBus;
                         addr <= OPB_ABus(15 downto 0);
             rnw <= OPB_RNW;
                   end if;
      end if;

-- register data read
   if OPB_Rst = '1' then
                  rdata <= X"0000_0000";
      elsif OPB_Clk'event and OPB_Clk ='1' then
                   if(ce0='1') then rdata(15 downto 0) <= bram_rdata;
end if;
                   if(ce1='1') then rdata(31 downto 16) <= bram_rdata;
end if;
      end if;

end process;

-- very important
-- TO DO
-- when writing, the read data can corrupe the DBus
      UIO_DBus <= rdata when xfer = '1' else X"0000_0000";


cs <= OPB_Select when OPB_ABus(31 downto 16)=X"FEFE" else '0';

-- the 1st ff -- FDR
process(OPB_Clk)
begin
      if OPB_Clk'event and OPB_Clk ='1' then
            if (xfer or xfer_1) = '1' then xfer <='0'; else xfer <= cs;
end if;
      end if;
end process;

process (OPB_Rst, OPB_Clk)
begin
      if OPB_Rst = '1' then
       xfer_1 <= '0';
```

```
             xfer_2 <= '0';
   elsif OPB_Clk'event and OPB_Clk ='1' then
             xfer_1 <= xfer;
             xfer_2 <= xfer_1 and not rnw;
      end if;
end process;

-- combinational logic for BRAM
we <= (xfer or xfer_1) and not rnw;
ce0 <= xfer_1 and not rnw;
ce1 <= xfer_2;
a0 <= xfer_1;
opb_ad_ce <= not xfer;

bram_addr <= (not audio_fifohalf) & addr(7 downto 2) & a0;
bram_wdata <= wdata(31 downto 16) when a0='0' else wdata(15 downto 0);

-- tie unused to ground
UIO_errAck <= '0';
UIO_retry <= '0';
UIO_toutSup <= '0';
UIO_xferAck <= xfer;


-- instantiate the BRAMS
process (bram_rdata_rev) begin
      for i in 0 to 15 loop
            bram_rdata(i)<=bram_rdata_rev(15-i);
      end loop;
end process;

process(bram_wdata) begin
      for i in 0 to 15 loop
            bram_wdata_rev(i)<=bram_wdata(15-i);
      end loop;
end process;

audio_bram_dacdata <= audio_bram_dacdata_v(0);
dac_bram : RAMB4_S1_S16 port map (
              DIA    => "0" ,
        ENA    => '1',
              WEA    => '0',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_dac,
              DOA    => audio_bram_dacdata_v,

              DIB    => bram_wdata_rev,
        ENB    => '1',
        WEB    => we,
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => open
);

audio_bram_adcdata_v(0) <= audio_bram_adcdata ;
```

```
adc_bram : RAMB4_S1_S16 port map (
              DIA    => audio_bram_adcdata_v,
        ENA    => '1',
              WEA    => '1',
        RSTA   => '0',
              CLKA   => audio_bram_clk,
              ADDRA  => audio_bram_addr_adc,
              DOA    => open,

              DIB    => X"0000",
        ENB    => '1',
        WEB    => '0',
              RSTB   => '0',
        CLKB   => OPB_Clk,
        ADDRB  => bram_addr,
        DOB    => bram_rdata_rev
);



-- the sound stuff

audio : audio_ak4565 port map(

      CLK => OPB_Clk,
   RST => OPB_Rst,

      audio_bram_clk => audio_bram_clk,
      audio_bram_addr_dac => audio_bram_addr_dac,
      audio_bram_addr_adc => audio_bram_addr_adc,
      audio_bram_dacdata =>   audio_bram_dacdata,
      audio_bram_adcdata => audio_bram_adcdata,

      interrupt => interrupt,
      audio_fifohalf => audio_fifohalf,

      AU_CSN_N => AU_CSN_N,
   AU_BCLK => AU_BCLK,
   AU_MCLK => AU_MCLK,
   AU_LRCK => AU_LRCK,
   AU_SDTI => AU_SDTI,
   AU_SDTO0 => AU_SDTO0
);

end Behavioral;
```

--Filename: fixed_16x16_multi.vhd

```
----------------------------------------------------------------------
--------
-- Fixed Point Multiplier
--
-- 16 bit data contains 9 bits to left of decimal and 7 to the right
-- 16 bit coeff contains all 16 bits to right of decimal
-- output has fixed point same as data
--
-- Josh Mackler
--
-- Modified by James Rishe, Andrew Melkonian and Narashiman
Chakravarthy
--
----------------------------------------------------------------------
--------



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity fixed_16x16_multi is
  port (
    data    : in  std_logic_vector(15 downto 0);
    coeff   : in  std_logic_vector(15 downto 0);
    rdy     : out std_logic;
    product : out std_logic_vector(15 downto 0);
    clk     : in std_logic;
    rst     : in std_logic;
    mode    : in std_logic_vector(1 downto 0));

end fixed_16x16_multi;

architecture archy of fixed_16x16_multi is

  signal tmp : std_logic_vector(15 downto 0);  -- shift amt of data
  signal multip : std_logic_vector(15 downto 0) := X"0000";
  signal inter1 : std_logic_vector(31 downto 0);
  signal inter2 : std_logic_vector(31 downto 0);
  signal inter3 : std_logic_vector(31 downto 0);
  signal inter4 : std_logic_vector(31 downto 0);
  signal inter5 : std_logic_vector(31 downto 0);
  signal inter6 : std_logic_vector(31 downto 0);
  signal inter7 : std_logic_vector(31 downto 0);
  signal inter8 : std_logic_vector(31 downto 0);
  signal inter9 : std_logic_vector(31 downto 0);
  signal inter10 : std_logic_vector(31 downto 0);
  signal inter11 : std_logic_vector(31 downto 0);
  signal inter12 : std_logic_vector(31 downto 0);
  signal inter13 : std_logic_vector(31 downto 0);
  signal inter14 : std_logic_vector(31 downto 0);
  signal inter15 : std_logic_vector(31 downto 0);
  signal second1, second2 : std_logic_vector(31 downto 0);
  signal res1 : std_logic_vector(31 downto 0);
  signal res2 : std_logic_vector(31 downto 0);
  signal res3 : std_logic_vector(31 downto 0);
  signal res4 : std_logic_vector(31 downto 0);
  signal res5 : std_logic_vector(31 downto 0);
  signal res6 : std_logic_vector(31 downto 0);
  signal res7 : std_logic_vector(31 downto 0);
  signal twos_coeff : std_logic_vector(15 downto 0);
 -- signal fm_count : std_logic_vector(0 to 3) := "0000";
  signal mul_pipe : std_logic_vector(1 downto 0);
  signal negative : std_logic := '0';
  signal neg1 : std_logic := '0';
  signal neg2 : std_logic := '0';
  signal go : std_logic := '0';
  signal tmp2 : std_logic_vector(15 downto 0);
  signal ready : std_logic := '0';
begin  -- archy


  inter1 <= "0" &tmp(15 downto 0) & X"000"&"000" when
twos_coeff(14)='1' and mode="00" else
            "0000000" & tmp(15 downto 7) &  X"0000" when
twos_coeff(7)='1' and mode="01" else X"00000000";

  inter2 <= "00" & tmp(15 downto 0)&X"000"&"00" when twos_coeff(13)='1'
and mode="00" else
            "000000" & tmp(15 downto 7) & '0'& X"0000" when
twos_coeff(8)='1'
            and mode = "01" else X"00000000";

  inter3 <= "000" & tmp(15 downto 0) & X"000" & "0"  when
twos_coeff(12)='1' and mode="00" else
            "00000" & tmp(15 downto 7) & "00" & X"0000" when
twos_coeff(9)='1'
            and mode = "01" else X"00000000";

  inter4 <= "0000" &  tmp(15 downto 0) & X"000" when twos_coeff(11)='1'
and mode="00" else
            "0000" & tmp(15 downto 7) & "000" & X"0000" when
twos_coeff(10)='1' and mode="01"
            else X"00000000";

  inter5 <=  "00000" & tmp(15 downto 0) & X"00" & "000" when
twos_coeff(10)='1' and mode="00" else
             "000" & tmp(15 downto 7) & X"00000" when
twos_coeff(11)='1' and mode="01"
             else X"00000000";

  inter6 <=  "000000" & tmp(15 downto 0) & X"00" & "00" when
twos_coeff(9)='1' and mode="00" else
             "00" & tmp(15 downto 7) & '0' & X"00000" when
twos_coeff(12)='1' and mode="01"
             else X"00000000";
  inter7 <=  "0000000" & tmp(15 downto 0) & X"00" & "0" when
twos_coeff(8)='1' and mode="00" else
             "0" & tmp(15 downto 7) & "00" & X"00000" when
twos_coeff(13)='1' and mode="01"
             else X"00000000";

  inter8 <=  "00000000" & tmp(15 downto 0) & X"00" when
twos_coeff(7)='1' and mode="00" else
             tmp(15 downto 7) & "000" & X"00000" when
twos_coeff(14)='1' and mode="01"
             else X"00000000";
  inter9 <=  "000000000" & tmp(15 downto 0) & "0000000" when
twos_coeff(6)='1' and mode="00" else X"00000000";
  inter10 <= "0000000000" & tmp(15 downto 0) & "000000"  when
twos_coeff(5)='1' and mode="00" else X"00000000";
  inter11 <= "00000000000" & tmp(15 downto 0) & "00000" when
twos_coeff(4)='1' and mode="00" else X"00000000";
  inter12 <= "000000000000" & tmp(15 downto 0) & "0000" when
twos_coeff(3)='1' and mode="00" else X"00000000";
  inter13 <= "0000000000000" & tmp(15 downto 0) & "000" when
twos_coeff(2)='1' and mode="00" else X"00000000";
  inter14 <= "00000000000000" & tmp(15 downto 0) & "00" when
twos_coeff(1)='1' and mode="00" else X"00000000";
  inter15 <= "000000000000000" & tmp(15 downto 0) & "0"  when
twos_coeff(0)='1' and mode="00" else X"00000000";

  process (clk, rst)
  begin  -- process
    if clk'event and clk='1' then
      if rst = '1' then
        if data(15) = '1' then
          tmp <= not(data) + 1;
          neg1 <= '1';
        else tmp <= data;
             neg1 <= '0';
        end if;
        if coeff(15) = '1' then
          neg2 <= '1';
          twos_coeff <= not(coeff) + 1;
        else
          twos_coeff <= coeff;
          neg2 <= '0';
        end if;
        multip <= X"0000";
        mul_pipe <= "00";
        ready <= '0';
        go <= '1';
      else
        if go = '1' then
          if mul_pipe = "00" then
            res1 <= inter1 + inter2;
            res2 <= inter3 + inter4;
            res3 <= inter5 + inter6;
            res4 <= inter7 + inter8;
            res5 <= inter9 + inter10;
            res6 <= inter11 + inter12;
            res7 <= inter13 + inter14;
            mul_pipe <= "01";

          elsif mul_pipe = "01" then       -- second clock
            second1 <= res1 + res2 + res3 + res4;
            second2 <= res5 + res6 + res7 + inter15;
            mul_pipe <= "11";
          elsif mul_pipe = "11" then       -- third clock
              if negative = '1' then product <= not(second1(31 downto
16) + second2(31 downto 16)) + 1;
            else
              product <= (second1(31 downto 16) + second2(31 downto
16));
            end if;
            go <= '0';
              ready <= '1';
          end if;
        end if;


      end if;
    end if;
  end process;

   rdy <= ready;
   negative <= neg1 xor neg2;
end archy;
```

--Filename: rom.vhd

```
---------------------
-- Marcio Buss     --
-- Anuj Maheshwari --
-- James Rishe     --
-- Andrew Melkonian--
---------------------
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity rom is
  port (
       clk  : in std_logic;
       en   : in std_logic;
       addr : in std_logic_vector(9 downto 0);
       data : out std_logic_vector(7 downto 0));
end rom;

architecture unknown of rom is
  type rom_type is array (0 to 1023) of std_logic_vector (7 downto 0);
  constant matrix : rom_type :=
(
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"7e", X"81", X"a5", X"81", X"bd", X"99", X"81", X"7e",
X"7e", X"ff", X"db", X"ff", X"c3", X"e7", X"ff", X"7e",
X"6c", X"fe", X"fe", X"fe", X"7c", X"38", X"10", X"00",
X"10", X"38", X"7c", X"fe", X"7c", X"38", X"10", X"00",
X"38", X"7c", X"38", X"fe", X"fe", X"d6", X"10", X"38",
X"10", X"38", X"7c", X"fe", X"fe", X"7c", X"10", X"38",
X"00", X"00", X"18", X"3c", X"3c", X"18", X"00", X"00",
X"ff", X"ff", X"e7", X"c3", X"c3", X"e7", X"ff", X"ff",
X"00", X"3c", X"66", X"42", X"42", X"66", X"3c", X"00",
X"ff", X"c3", X"99", X"bd", X"bd", X"99", X"c3", X"ff",
X"0f", X"07", X"0f", X"7d", X"cc", X"cc", X"cc", X"78",
X"3c", X"66", X"66", X"66", X"3c", X"18", X"7e", X"18",
X"3f", X"33", X"3f", X"30", X"30", X"70", X"f0", X"e0",
X"7f", X"63", X"7f", X"63", X"63", X"67", X"e6", X"c0",
X"18", X"db", X"3c", X"e7", X"e7", X"3c", X"db", X"18",
X"80", X"e0", X"f8", X"fe", X"f8", X"e0", X"80", X"00",
X"02", X"0e", X"3e", X"fe", X"3e", X"0e", X"02", X"00",
X"18", X"3c", X"7e", X"18", X"18", X"7e", X"3c", X"18",
X"66", X"66", X"66", X"66", X"66", X"00", X"66", X"00",
X"7f", X"db", X"db", X"7b", X"1b", X"1b", X"1b", X"00",
X"3e", X"61", X"3c", X"66", X"66", X"3c", X"86", X"7c",
X"00", X"00", X"00", X"00", X"7e", X"7e", X"7e", X"00",
X"18", X"3c", X"7e", X"18", X"7e", X"3c", X"18", X"ff",
X"18", X"3c", X"7e", X"18", X"18", X"18", X"18", X"00",
X"18", X"18", X"18", X"18", X"7e", X"3c", X"18", X"00",
X"00", X"18", X"0c", X"fe", X"0c", X"18", X"00", X"00",
X"00", X"30", X"60", X"fe", X"60", X"30", X"00", X"00",
X"00", X"00", X"c0", X"c0", X"c0", X"fe", X"00", X"00",
X"00", X"24", X"66", X"ff", X"66", X"24", X"00", X"00",
X"00", X"18", X"3c", X"7e", X"ff", X"ff", X"00", X"00",
X"00", X"ff", X"ff", X"7e", X"3c", X"18", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"00",
X"18", X"3c", X"3c", X"18", X"18", X"00", X"18", X"00",
X"66", X"66", X"24", X"00", X"00", X"00", X"00", X"00",
X"0c", X"6e", X"7c", X"ee", X"7c", X"ec", X"60", X"00",
X"18", X"3e", X"60", X"3c", X"06", X"7c", X"18", X"00",
X"00", X"c6", X"cc", X"18", X"30", X"66", X"c6", X"00",
X"38", X"6c", X"38", X"76", X"dc", X"cc", X"76", X"00",
X"18", X"18", X"30", X"00", X"00", X"00", X"00", X"00",
X"0c", X"18", X"30", X"30", X"30", X"18", X"0c", X"00",
X"30", X"18", X"0c", X"0c", X"0c", X"18", X"30", X"00",
X"00", X"66", X"3c", X"ff", X"3c", X"66", X"00", X"00",
X"00", X"18", X"18", X"7e", X"18", X"18", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"18", X"18", X"30",
X"00", X"00", X"00", X"7e", X"00", X"00", X"00", X"00",
X"00", X"00", X"00", X"00", X"00", X"18", X"18", X"00",
X"06", X"0c", X"18", X"30", X"60", X"c0", X"80", X"00",
X"38", X"6c", X"c6", X"d6", X"c6", X"6c", X"38", X"00",
X"18", X"38", X"18", X"18", X"18", X"18", X"7e", X"00",
X"7c", X"c6", X"06", X"1c", X"30", X"66", X"fe", X"00",
X"7c", X"c6", X"06", X"3c", X"06", X"c6", X"7c", X"00",
X"1c", X"3c", X"6c", X"cc", X"fe", X"0c", X"1e", X"00",
X"fe", X"c0", X"c0", X"fc", X"06", X"c6", X"7c", X"00",
X"38", X"60", X"c0", X"fc", X"c6", X"c6", X"7c", X"00",
X"fe", X"c6", X"0c", X"18", X"30", X"30", X"30", X"00",
X"7c", X"c6", X"c6", X"7c", X"c6", X"c6", X"7c", X"00",
X"7c", X"c6", X"c6", X"7e", X"06", X"0c", X"78", X"00",
X"00", X"18", X"18", X"00", X"00", X"18", X"18", X"00",
X"00", X"18", X"18", X"00", X"00", X"18", X"18", X"30",
X"06", X"0c", X"18", X"30", X"18", X"0c", X"06", X"00",
X"00", X"00", X"7e", X"00", X"00", X"7e", X"00", X"00",
X"60", X"30", X"18", X"0c", X"18", X"30", X"60", X"00",
X"7c", X"c6", X"0c", X"18", X"18", X"00", X"18", X"00",
X"7c", X"c6", X"de", X"de", X"de", X"c0", X"78", X"00",
X"38", X"6c", X"c6", X"fe", X"c6", X"c6", X"c6", X"00",
X"fc", X"66", X"66", X"7c", X"66", X"66", X"fc", X"00",
X"3c", X"66", X"c0", X"c0", X"c0", X"66", X"3c", X"00",
X"f8", X"6c", X"66", X"66", X"66", X"6c", X"f8", X"00",
X"fe", X"62", X"68", X"78", X"68", X"62", X"fe", X"00",
X"fe", X"62", X"68", X"78", X"68", X"60", X"f0", X"00",
X"3c", X"66", X"c0", X"c0", X"ce", X"66", X"3a", X"00",
X"c6", X"c6", X"c6", X"fe", X"c6", X"c6", X"c6", X"00",
X"3c", X"18", X"18", X"18", X"18", X"18", X"3c", X"00",
X"1e", X"0c", X"0c", X"0c", X"cc", X"cc", X"78", X"00",
X"e6", X"66", X"6c", X"78", X"6c", X"66", X"e6", X"00",
X"f0", X"60", X"60", X"60", X"62", X"66", X"fe", X"00",
X"c6", X"ee", X"fe", X"fe", X"d6", X"c6", X"c6", X"00",
X"c6", X"e6", X"f6", X"de", X"ce", X"c6", X"c6", X"00",
X"7c", X"c6", X"c6", X"c6", X"c6", X"c6", X"7c", X"00",
X"fc", X"66", X"66", X"7c", X"60", X"60", X"f0", X"00",
X"7c", X"c6", X"c6", X"c6", X"c6", X"ce", X"7c", X"0e",
X"fc", X"66", X"66", X"7c", X"6c", X"66", X"e6", X"00",
X"3c", X"66", X"30", X"18", X"0c", X"66", X"3c", X"00",
X"7e", X"7e", X"5a", X"18", X"18", X"18", X"3c", X"00",
X"c6", X"c6", X"c6", X"c6", X"c6", X"c6", X"7c", X"00",
X"c6", X"c6", X"c6", X"c6", X"c6", X"6c", X"38", X"00",
X"c6", X"c6", X"c6", X"d6", X"d6", X"fe", X"6c", X"00",
X"c6", X"c6", X"6c", X"38", X"6c", X"c6", X"c6", X"00",
X"66", X"66", X"66", X"3c", X"18", X"18", X"3c", X"00",
X"fe", X"c6", X"8c", X"18", X"32", X"66", X"fe", X"00",
X"3c", X"30", X"30", X"30", X"30", X"30", X"3c", X"00",
X"c0", X"60", X"30", X"18", X"0c", X"06", X"02", X"00",
X"3c", X"0c", X"0c", X"0c", X"0c", X"0c", X"3c", X"00",
X"c0", X"c0", X"dc", X"e2", X"c2", X"cc", X"f0", X"00",
X"00", X"00", X"00", X"00", X"00", X"00", X"00", X"ff",
X"30", X"18", X"0c", X"00", X"00", X"00", X"00", X"00",
X"00", X"00", X"78", X"0c", X"7c", X"cc", X"76", X"00",
X"e0", X"60", X"7c", X"66", X"66", X"66", X"dc", X"00",
X"00", X"00", X"7c", X"c6", X"c0", X"c6", X"7c", X"00",
X"1c", X"0c", X"7c", X"cc", X"cc", X"cc", X"76", X"00",
X"00", X"00", X"7c", X"c6", X"fe", X"c0", X"7c", X"00",
X"3c", X"66", X"60", X"f8", X"60", X"60", X"f0", X"00",
X"00", X"00", X"76", X"cc", X"cc", X"7c", X"0c", X"f8",
X"e0", X"60", X"6c", X"76", X"66", X"66", X"e6", X"00",
X"18", X"00", X"38", X"18", X"18", X"18", X"3c", X"00",
X"06", X"00", X"06", X"06", X"06", X"66", X"66", X"3c",
X"e0", X"60", X"66", X"6c", X"78", X"6c", X"e6", X"00",
X"38", X"18", X"18", X"18", X"18", X"18", X"3c", X"00",
X"00", X"00", X"ec", X"fe", X"d6", X"d6", X"d6", X"00",
X"00", X"00", X"dc", X"66", X"66", X"66", X"66", X"00",
X"00", X"00", X"7c", X"c6", X"c6", X"c6", X"7c", X"00",
X"00", X"00", X"dc", X"66", X"66", X"7c", X"60", X"f0",
X"00", X"00", X"76", X"cc", X"cc", X"7c", X"0c", X"1e",
X"00", X"00", X"dc", X"76", X"60", X"60", X"f0", X"00",
X"00", X"00", X"7e", X"c0", X"7c", X"06", X"fc", X"00",
X"30", X"30", X"fc", X"30", X"30", X"36", X"1c", X"00",
X"00", X"00", X"cc", X"cc", X"cc", X"cc", X"76", X"00",
X"00", X"00", X"c6", X"c6", X"c6", X"6c", X"38", X"00",
X"00", X"00", X"c6", X"d6", X"d6", X"fe", X"6c", X"00",
X"00", X"00", X"c6", X"6c", X"38", X"6c", X"c6", X"00",
X"00", X"00", X"c6", X"c6", X"c6", X"7e", X"06", X"fc",
X"00", X"00", X"7e", X"4c", X"18", X"32", X"7e", X"00",
X"0e", X"18", X"18", X"70", X"18", X"18", X"0e", X"00",
X"18", X"18", X"18", X"18", X"18", X"18", X"18", X"00",
X"70", X"18", X"18", X"0e", X"18", X"18", X"70", X"00",
X"76", X"dc", X"00", X"00", X"00", X"00", X"00", X"00",
X"00", X"10", X"38", X"6c", X"c6", X"c6", X"fe", X"00");


  begin
    process (clk)
      begin
        if (clk'event and clk='1') then
          if en = '1' then
            data <= matrix(conv_integer(addr));
          end if;
        end if;
    end process;
  end unknown;
```

10. C Code

//Filename: main.c

```
// main.c, created by James Rishe and Andrew Melkonian
//  using main.c from Nortsam as a basis

#include "xparameters.h"
#include "xbasic_types.h"
#include "xio.h"
#include "xintc_l.h"
#include "xuartlite_l.h"
#include "fft_settings.h"

#define FONT_OFFSET 0xA00

#define RAM_H 80
#define RAM_V 60

#define VGA_START 0x00800000
#define AUDIO_START 0xFEFE0000
#define LAST_LINE 59

#define _A 1
#define _B 2
#define _C 3
#define _D 4
#define _E 5
#define _F 6
#define _G 7
#define _0 8
#define _1 9
#define _2 10
#define _3 11
#define _4 12
#define _5 13
#define _6 14
#define _7 15
#define _8 16
#define _9 17
#define _m 18
#define _M 19
#define _P 20
#define _s 21
#define _fl 22
#define XPAR_INTC_AK4565_INTERRUPT_INTR 0


// defined in isr.c
extern void audio_intr_handler(void *callback);
extern volatile Xuint16 bufInUse[];
extern volatile Xuint16 bufReady[];

Xint32 audioBuffer[ FFT_SIZE << 1 ];
Xint16 audioBufferLEFT[ FFT_SIZE << 1 ];
```

```
Xint32 *astart;
Xint16 *startLPtr, *startRPtr;


/*
 * setup_interrupts: Initialize the interrupt sources and handlers
 */
void setup_interrupts()
{
  /*
   * Reset the interrupt controller peripheral
   */

  /* Disable the interrupt signal */
  XIntc_mMasterDisable(XPAR_INTC_SINGLE_BASEADDR);
  /* Disable all interrupt sources */
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,0);
  /* Acknowledge all possible interrupt sources
     to make sure none are pending */
  XIntc_mAckIntr(XPAR_INTC_SINGLE_BASEADDR, 0xffffffff);

  /*
   * Install the codec interrupt handler
   */
  XIntc_InterruptVectorTable[XPAR_INTC_AK4565_INTERRUPT_INTR].Handler =
    audio_intr_handler;

  // enable interrupt sources
  /* Enable CPU interrupts */
  microblaze_enable_interrupts();
  /* Enable interrupts from the interrupt controller */
  XIntc_mMasterEnable(XPAR_INTC_SINGLE_BASEADDR);
  /* Tell the interrupt controller to accept interrupts from the codec
*/
  XIntc_mEnableIntr(XPAR_INTC_SINGLE_BASEADDR,
XPAR_AK4565_INTERRUPT_MASK);
}



/* Address of a particular character on the screen (rows are 80) */
#define CHAR(r,c) \
   (((unsigned char *)(XPAR_XSB300_BASEADDR))[(c) + ((r) << 6) + ((r)
<< 4)])

/* Start of font memory */
#define FONT ((unsigned char *)XPAR_XSB300_BASEADDR + FONT_OFFSET)

/* Cursor position on the screen */
int x = 0;
int y = 0;

/* Place a single character ch on the screen at position (column, row)
*/
void put_char(char ch, int row, int column)
{
  CHAR(row, column) = ch;
```

```
}


int main()
{
  int i;
  int j,k,l;
  Xint32 maxmag, maxind;
  volatile unsigned char *p;
  Xuint32 data;
  // Enable the instruction cache: makes the code run 6 times faster
  microblaze_enable_icache();

  setup_interrupts();
  l=0;

  // Clear  the  relevant //
  // positions of the RAM //
  // i.e., the first 4800 //
  for(x = 0 ; x < RAM_H*RAM_V; x++)
    XIo_Out8(VGA_START+x, 0);

  // put static words on the screen
  setup_graphics();

  // this never changes, define outside loop.
  startLPtr = (Xint16*)(audioBufferLEFT)-1;


  i = 0;
  for ( ;; ) {
    // ******************************
    // double buffer logistics

    while (bufReady[i] == 0)
      print(""); // need this delay, a busy-wait loop prevents isr
context switch?!
    bufInUse[(i+1) % 2] = 0;
    bufInUse[i] = 1;

    // fudge starting pointers to account for FFT starting
    //  with index 1.
    astart = (Xint32*)(&(audioBuffer[ (i<<BUFFER_SHIFT_OFFSET) ])) - 1;


    separate(astart, startLPtr);
    //    putnum(startLPtr[5]);



    bitReverse( startLPtr );
    //    putnum(startLPtr[5]);
    // after this point:
    //   audioBuffer[] holds bit-rearranged LEFT samples,
MSB=real/LSB=imag
```

```
    doFFT( startLPtr );
    /*    maxind = 0;
    maxmag = 0;
    for(i = 0; i < FFT_SIZE; i++){
      if(startLPtr[i] > maxmag){
      maxind = i;
      maxmag = startLPtr[i];
      }
    }
    print("\r\nMax value: ");
    putnum(maxmag);
    print("  Max index:");
    putnum(maxind);*/
    //    if(l == 15){
    visualize(startLPtr);
    //      l = 0;
    //    }
    l++;
    i = (i+1) & 1;
  }

  print("Goodbye\r\n");
  return 0;
}
```

//Filename: fft.c

```
// fft.c, created by James Rishe and Andrew Melkonian
// using examples from team Nortsam

#include "xbasic_types.h"
#include "xio.h"
#include "coeff1024.c"  // coefficients must be calculated for FFT size
#include "fft_settings.h"

//extern Xuint32 coeff[];


void separate(Xint32* adata, Xint16 *dataL ) {
  int i,j;
  Xint16* aiter, *aend;
  Xint16* diter, *dend;

  aiter = adata+1;
  diter = dataL+1;
  aend = adata+FFT_SIZE;
  dend = dataL+(FFT_SIZE<<1);

  for(i = 1,j=1 ; i <= FFT_SIZE;++i, j+=2 ){
    dataL[j] = (Xint16)((adata[i] & 0xFFFF0000) >> 16);
    //    dataL[j] <<=1; //amplify
    dataL[j+1] = 0x0000;
    //dataR[i] = (Xint16)(adata[i] & 0x0000FFFF);
  }
}

/**
 * takes buffer data in dataL, in form 0xLLLLRRRR
 * separates L and R data and performs bit reversal of indices
 * returns two arrays, each ready to be FFT-ized
 * ---note: as with the FFT algorithm, this function will never access
idx[0]
 * fills all imaginary (even) indices with 0x0000.
 */
void bitReverse( Xint16* data ){
  int i, j, n, m;
  Xint16 temp;

  n = FFT_SIZE << 1;
  j = 1;

  for ( i=1; i <= n; i+=2 ) {
    if ( j > i ) {

      temp = data[i];
      data[i] = data[j];
      data[j] = temp;

      /*  Not necessary because these are
        zero-valued
      temp = data[i+1];
      data[i+1] = data[j+1];
      data[j+1] = temp; */

    }

    m = FFT_SIZE;
    while ( (m >= 2) && (j > m) ) {
      j -= m;
      m >>= 1;
    }

    j += m;
  }

}

// takes an array of size (FFT_SIZE * 2), of 16-bit values
// this function WILL NEVER access array index 0 (very important-no seg
faults)
void doFFT( Xint16 *data ) {
  //  int istep, mmax, j, i, m, n, coeff_ofs, z;
  Xint32 istep, mmax, j, i, m, n;
  Xuint32* coeffp;

  Xint16* tempRI;
  Xuint32 temp=0;
  Xint16 tempr;
  Xint16 tempi;

  coeffp = coeff;

  n = FFT_SIZE << 1;
  mmax = 2;
  while ( n > mmax ) { // this, the outer loop, gets executed
log2(FFT_SIZE) times.
    istep = mmax << 1;
    //    print("");
    for ( m=1; m<mmax; m+=2 ) {
      for ( i=m; i<=n; i+=istep ) {
      j = i + mmax;

      // The Danielson-Lanczos formula, embedded systems
      // style
      //Write data to the multiplier (no imaginary part)
      XIo_Out32( 0x0FEF0008, ((Xuint32)data[j])<<16);
      //Write coefficient to the multiplier
      XIo_Out32( 0x0FEF000C, *coeffp );
      temp = XIo_In32( 0x0FEF0000 );

      //    tempr = (Xint16)(temp >> 16);
      //    tempi = (Xint16)(temp & 0x0000FFFF);
      tempRI = (Xint16*)(&temp);

      data[j] = data[i] - tempRI[0];
      data[j+1] = data[i+1] - (tempRI[1]);
      data[i] += tempRI[0];
      data[i+1] += tempRI[1];
      /*
      data[j] = data[i] - (Xint16)(temp>>16);
      data[j+i] = data[i+1] - temp&0x0000FFFF;
      data[i] += (Xint16)(temp>>16);
      data[i+1] += temp&0x0000FFFF;
      */
      }
      coeffp++;
    }
    mmax = istep;
    /*    putnum(n);
    print("$");
    putnum(mmax);
    print("$\n\r");*/


  }
  //  print("FFT DONE \n\r");
  //print("Going back now");
}
```

//Filename: isr.c

```
// isr.c, created by Nortsam

#include "xbasic_types.h"
#include "xio.h"
#include "xparameters.h"
#include "xuartlite_l.h"
#include "fft_settings.h"

extern Xuint32 audioBuffer[];
extern Xuint32 audioBufferRIGHT[];
Xint32 bufPos = 0;
Xuint16 volatile bufInUse[] = {0, 0};
Xuint16 volatile bufReady[] = {0, 0};

// define: packet = set of 64 samples; frame= set of FFT_SIZE samples

/*
Xuint32 droppedPackets=0;
Xuint32 grabbedPackets=0;
Xuint32 midFrameDrops=0;
*/

//#define FFT_SIZE 2048

/*
 * Interrupt service routine for the AK4565 CODEC
 */

void audio_intr_handler( void *callback ) {
  Xint16 i;

  // if the FFT isn't done yet, drop these samples.
  if (bufPos < FFT_SIZE) {
    if (bufInUse[0] == 1) {
      /*
      droppedPackets++;
      if (bufPos != 0)
      midFrameDrops++;
      */
      return;
    }
  }
  else if (bufInUse[1] == 1) {

    //    droppedPackets++;
    //      midFrameDrops++;
    if (bufPos != FFT_SIZE)
      return;
  }

  for (i=0; i < 64; i++) {
    XIo_Out32( XPAR_AK4565_BASEADDR + (i<<2), 0 );
    audioBuffer[bufPos++] = (XIo_In32( XPAR_AK4565_BASEADDR )); //
0xLLLLRRRR
```

```
  }

  if (bufPos == FFT_SIZE) {
    bufReady[0] = 1;
    bufReady[1] = 0;
    //putnum(i);
    //print("\r\n");
  }
  else if (bufPos == (FFT_SIZE << 1)) { // multiply here OK -- should
optimize to constant value
    bufReady[1] = 1;
    bufReady[0] = 0;
    bufPos = 0;
    //    putnum(audioBuffer[12]);
    //print("\r\n");
    //putnum(i+1);
    //print("\r\n");


  }

  //  grabbedPackets++;
}
```

//Filename: graphics.c

```
// graphics.c, created by James Rishe, Andrew Melkonian
//   Narashiman Chakravarthy
// Using some functionality from Nortsam

#include "xbasic_types.h"
#include "xio.h"

#include "xintc_l.h"
#include "xuartlite_l.h"
#include "fft_settings.h"

#define W 640
#define H 480
#define VGA_START 0x00800000
#define X_START 40
#define Y_START 240

#define RED 0xE0
#define GREEN 0x1C
#define BLUE 0x03
#define WHITE RED | GREEN | BLUE
#define BLACK 0x00

#define MAX_PIX 200

#define TRUE 1
#define FALSE 0

/*#define CHAR(r,c) \
  ([(c) + ((r) << 6) + ((r) << 4)])*/
#define CHAR(r,c) \
   (((unsigned char *)(VGA_START))[(c) + ((r) << 6) + ((r) << 4)])

void put_string(char *string, int row, int column)
{
  char *p = &(CHAR(row, column));
  while (*p++ = *string++)
    ;
}


Xuint16 correspond[] = {
     0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
    10,   11,   13,   16,   20,   25,   31,   39,   49,   61,
    77,   97,  122,  153,  194,  244,  306,  387,  487,  612,
   773,  973};
/*********************************************************************/
Xuint16 range[] = {
  //    1     2     3      4      5      6      7      8      9
  0,    0,    1,    1,     1,     2,     2,     2,     2,     2,
  2,    3,    3,    3,     3,     3,     3,     3,     3,     3,
  3,    3,    3,    3,     4,     4,     4,     4,     4,     4,
  4,    4,    4,    4,     4,     4,     4,     4,     4,     4,
  4,    4,    4,    4,     4,     4,     4,     4,     5,     5,
  5,    5,    5,    5,     5,     5,     5,     5,     5,     5,
  5,    5,    5,    5,     5,     5,     5,     5,     5,     5,
  5,    5,    5,    5,     5,     5,     5,     5,     5,     5,
  5,    5,    5,    5,     5,     5,     5,     5,     5,     5,
  5,    5,    5,    5,     5,     5,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
  6,    6,    6,    6,     6,     6,     6,     6,     6,     6,
};

int lastmax;



Xuint16 calcMagnitude( Xuint16 datar, Xuint16 datai ) {
  Xint16 *tempRI;
  Xuint32 result;
  short high = 182, low = 0, middle;
  unsigned char less = 0;
  short temp;

  // calculate magnitudes of buckets corresponding to positive
frequencies
  // hw address 0x0fef_0014 squares real and imag.

  XIo_Out32( 0x0FEF0014, (Xuint32)(datar)<<16 | (Xuint32)datai  );
  result = XIo_In32( 0x0FEF0000 );

  tempRI = (Xuint16*)(&result);
  result = (Xuint32)(tempRI[0]) + (Xuint32)(tempRI[1]);

  // saturate, don't overflow
  if (result > 32767)
    result = 0;
  return result;
}

const int startrow = 13;
const int col = 25;
const int inc = 3;

void setup_graphics(){
  int sr = startrow;
  put_string("The B^ B# Pitch Detector", 2, 3);
  put_string("Very High pitch", sr, col);
  put_string("High pitch", sr+=inc, col);
  put_string("Mid-high pitch", sr+=inc, col);
  put_string("Mid pitch", sr+=inc, col);
  put_string("Mid-low pitch", sr+=inc, col);
  put_string("Low pitch", sr+=inc, col);
  put_string("No sound", sr+=inc, col);

}
```

```
/* Wipe the screen in any places that are dynamically
   written
*/
void wipe(){
  int c = col - 2;
  int i;
  int incx5 = (inc << 2 + inc);
  for(i = startrow; i < startrow+incx5; i+= inc){
    CHAR(i, c) = ' ';
  }
}

void showgraphics(Xint16 ind){
  int i;
  int a = 6-ind;
  int nincs = inc*a;
  wipe();
  CHAR(startrow+nincs, col-2) = 13;

}

int tally[] = {0, 0, 0, 0, 0, 0, 0};
int iternum = 0;
void visualize(Xint16* data){
  Xint16 maxmag, maxind, k, i;
  maxmag = maxind = 0;

  for(k = 1; k < 340; k+=2){
    if(calcMagnitude(data[k], data[k+1]) > maxmag){
      maxind = k;
      maxmag = calcMagnitude(data[k], data[k+1]);
    }
  }

  tally[range[maxind>>1]]++;
  iternum++;
  if(iternum == 10){
    maxind = 0;
    maxmag = 1;
    for(i = 1; i < 7; i++){
      if(tally[i] > maxmag){
      maxind = i;
      maxmag = tally[i];
      }
      tally[i] = 0;
    }
    showgraphics(maxind);
    iternum = 0;
  }
}
```

// Filename: fft_settings.h

```
// fft_settings.h, created by James Rishe

#ifndef _FFT_SETTINGS
#define _FFT_SETTINGS

#define FFT_SIZE 1024
#define BUFFER_SHIFT_OFFSET 10

#endif
```

11. Other files

# Filename: system.mhs

```
# Parameters
PARAMETER VERSION = 2.0.0

# Global Ports

PORT PB_A = PB_A, DIR = OUT, VEC = [19:0]
PORT PB_D = PB_D, DIR = INOUT, VEC = [15:0]
PORT PB_LB_N = PB_LB_N, DIR = OUT
PORT PB_UB_N = PB_UB_N, DIR = OUT
PORT PB_WE_N = PB_WE_N, DIR = OUT
PORT PB_OE_N = PB_OE_N, DIR = OUT
PORT RAM_CE_N = RAM_CE_N, DIR = OUT
PORT VIDOUT_CLK = VIDOUT_CLK, DIR = OUT
PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N, DIR = OUT
PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N, DIR = OUT
PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N, DIR = OUT
PORT VIDOUT_RCR = VIDOUT_RCR, DIR = OUT, VEC = [9:0]
PORT VIDOUT_GY = VIDOUT_GY, DIR = OUT, VEC = [9:0]
PORT VIDOUT_BCB = VIDOUT_BCB, DIR = OUT, VEC = [9:0]
PORT FPGA_CLK1 = FPGA_CLK1, DIR = IN
PORT RS232_TD = RS232_TD, DIR=OUT
PORT RS232_RD = RS232_RD, DIR=IN
PORT AU_CSN_N =  AU_CSN_N, DIR=OUT
PORT AU_BCLK =  AU_BCLK, DIR=OUT
PORT AU_MCLK = AU_MCLK, DIR=OUT
PORT AU_LRCK = AU_LRCK, DIR=OUT
PORT AU_SDTI = AU_SDTI, DIR=OUT
PORT AU_SDTO0 = AU_SDTO0, DIR=IN

# Sub Components


BEGIN microblaze
 PARAMETER INSTANCE = mymicroblaze
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_USE_BARREL = 1
 PARAMETER C_USE_ICACHE = 0
# PARAMETER C_ADDR_TAG_BITS = 6
# PARAMETER C_CACHE_BYTE_SIZE = 2048
# PARAMETER C_ICACHE_BASEADDR = 0x00860000
# PARAMETER C_ICACHE_HIGHADDR = 0x0087FFFF
 PORT Clk = sys_clk
 PORT Reset = fpga_reset
 PORT Interrupt = intr
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE DOPB = myopb_bus
 BUS_INTERFACE IOPB = myopb_bus
END

BEGIN opb_intc
```

```
 PARAMETER INSTANCE = intc
 PARAMETER HW_VER = 1.00.c
 PARAMETER C_BASEADDR = 0xFFFF0000
 PARAMETER C_HIGHADDR = 0xFFFF00FF
 PORT OPB_Clk = sys_clk
 PORT Intr =  uart_intr & audio_intr
 PORT Irq =   intr
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN bram_block
 PARAMETER INSTANCE = bram
 PARAMETER HW_VER = 1.00.a
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_xsb300
 PARAMETER INSTANCE = xsb300
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00800000
 PARAMETER C_HIGHADDR = 0x00FFFFFF
 PORT PB_A = PB_A
 PORT PB_D = PB_D
 PORT PB_LB_N = PB_LB_N
 PORT PB_UB_N = PB_UB_N
 PORT PB_WE_N = PB_WE_N
 PORT PB_OE_N = PB_OE_N
 PORT RAM_CE_N = RAM_CE_N
 PORT OPB_Clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT VIDOUT_CLK = VIDOUT_CLK
 PORT VIDOUT_HSYNC_N = VIDOUT_HSYNC_N
 PORT VIDOUT_VSYNC_N = VIDOUT_VSYNC_N
 PORT VIDOUT_BLANK_N = VIDOUT_BLANK_N
 PORT VIDOUT_RCR = VIDOUT_RCR
 PORT VIDOUT_GY = VIDOUT_GY
 PORT VIDOUT_BCB = VIDOUT_BCB
 BUS_INTERFACE SOPB = myopb_bus
END

BEGIN clkgen
 PARAMETER INSTANCE = clkgen_0
 PARAMETER HW_VER = 1.00.a
 PORT FPGA_CLK1 = FPGA_CLK1
 PORT sys_clk = sys_clk
 PORT pixel_clock = pixel_clock
 PORT fpga_reset = fpga_reset
END

BEGIN lmb_lmb_bram_if_cntlr
 PARAMETER INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x00000000
 PARAMETER C_HIGHADDR = 0x00000FFF
 BUS_INTERFACE DLMB = d_lmb
 BUS_INTERFACE ILMB = i_lmb
 BUS_INTERFACE PORTA = conn_0
 BUS_INTERFACE PORTB = conn_1
END

BEGIN opb_uartlite
 PARAMETER INSTANCE = myuart
 PARAMETER HW_VER = 1.00.b
 PARAMETER C_CLK_FREQ = 50_000_000
 PARAMETER C_USE_PARITY = 0
 PARAMETER C_BASEADDR = 0xFEFF0100
 PARAMETER C_HIGHADDR = 0xFEFF01FF
 PORT OPB_Clk = sys_clk
 PORT Interrupt = uart_intr
 BUS_INTERFACE SOPB = myopb_bus
 PORT RX=RS232_RD
 PORT TX=RS232_TD
END

BEGIN opb_v20
 PARAMETER INSTANCE = myopb_bus
 PARAMETER HW_VER = 1.10.a
 PARAMETER C_DYNAM_PRIORITY = 0
 PARAMETER C_REG_GRANTS = 0
 PARAMETER C_PARK = 0
 PARAMETER C_PROC_INTRFCE = 0
 PARAMETER C_DEV_BLK_ID = 0
 PARAMETER C_DEV_MIR_ENABLE = 0
 PARAMETER C_BASEADDR = 0x0fff1000
 PARAMETER C_HIGHADDR = 0x0fff10ff
 PORT SYS_Rst = fpga_reset
 PORT OPB_Clk = sys_clk
END

BEGIN lmb_v10
 PARAMETER INSTANCE = d_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN lmb_v10
 PARAMETER INSTANCE = i_lmb
 PARAMETER HW_VER = 1.00.a
 PORT LMB_Clk = sys_clk
 PORT SYS_Rst = fpga_reset
END

BEGIN opb_xsb300_ak4565
 PARAMETER INSTANCE = ak4565
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xFEFE0000
 PARAMETER C_HIGHADDR = 0xFEFEFFFF

 PORT OPB_Clk = sys_clk
 PORT Interrupt = audio_intr
 PORT AU_CSN_N =  AU_CSN_N
 PORT AU_BCLK =  AU_BCLK
 PORT AU_MCLK = AU_MCLK
 PORT AU_LRCK = AU_LRCK
 PORT AU_SDTI = AU_SDTI
 PORT AU_SDTO0 = AU_SDTO0

 BUS_INTERFACE SOPB = myopb_bus
END


BEGIN opb_xsb300_nortsam
 PARAMETER INSTANCE = nortsam
 PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0x0FEF0000
 PARAMETER C_HIGHADDR = 0x0FEFFFFF

 PORT OPB_Clk = sys_clk
 PORT Interrupt = nortsam_intr

 BUS_INTERFACE SOPB = myopb_bus
END
```

#Filename: system.mss

```
PARAMETER VERSION = 2.0.0
PARAMETER HW_SPEC_FILE = system.mhs

BEGIN PROCESSOR
 PARAMETER HW_INSTANCE = mymicroblaze
 PARAMETER DRIVER_NAME = cpu
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER EXECUTABLE = system.elf
 PARAMETER COMPILER = microblaze-gcc
 PARAMETER ARCHIVER = microblaze-ar
 PARAMETER DEFAULT_INIT = EXECUTABLE
 PARAMETER STDIN = myuart
 PARAMETER STDOUT = myuart
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = xsb300
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = lmb_lmb_bram_if_cntlr_0
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = myuart
 PARAMETER DRIVER_NAME = uartlite
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END

BEGIN DRIVER
 PARAMETER HW_INSTANCE = intc
 PARAMETER DRIVER_NAME = intc
 PARAMETER DRIVER_VER = 1.00.b
 PARAMETER LEVEL = 0
END
```